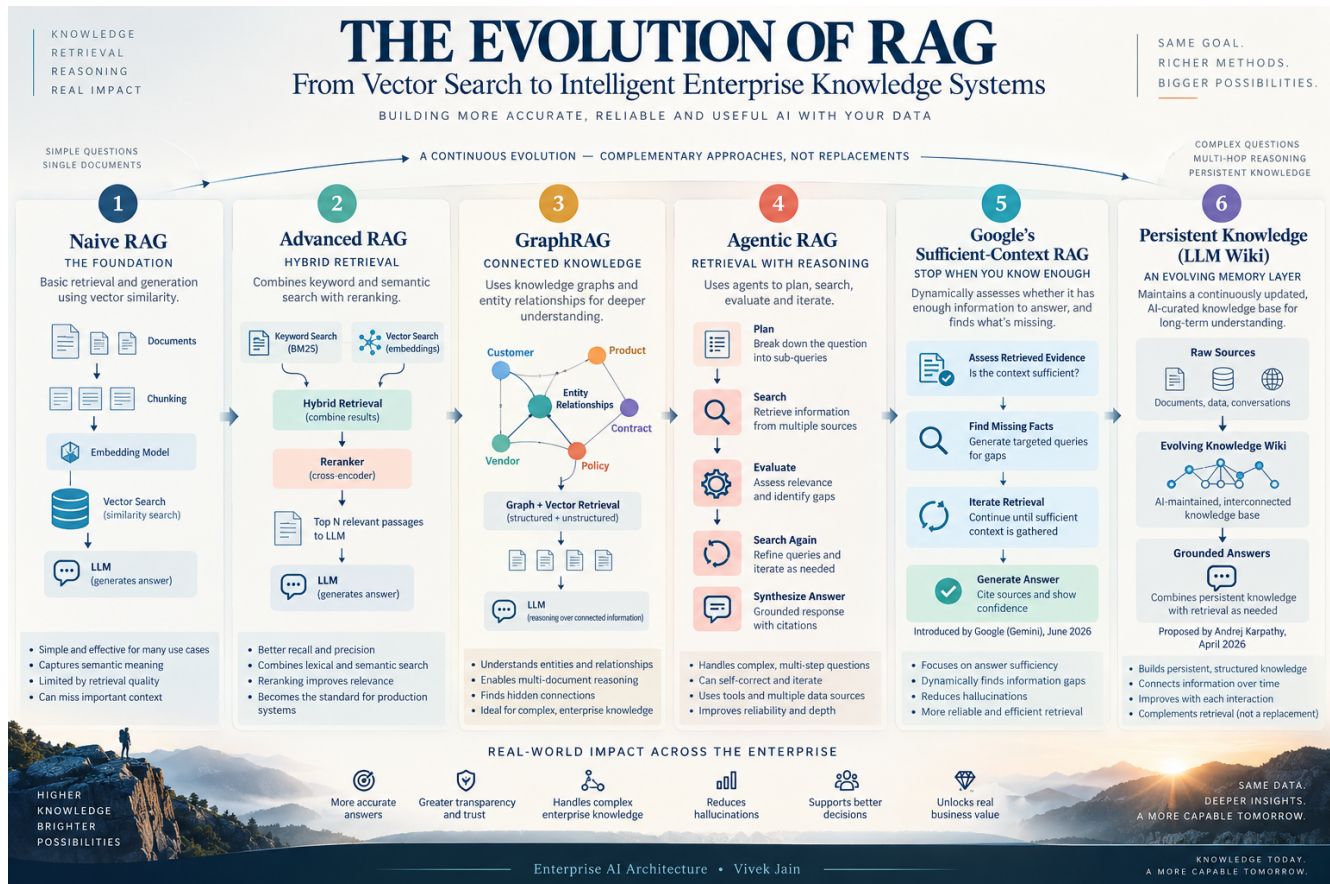


The Evolution of RAG

From Vector Search to Intelligent Enterprise Knowledge Systems



Publication draft · October 12, 2026 · Case-study facts and disclosure permissions require approval.

How document retrieval grew into hybrid search, graph reasoning, iterative agents, and persistent knowledge—and what it means for enterprise AI teams.

By Vivek Jain · Enterprise AI Architecture & Transformation

The question that changes everything

Imagine an employee responding to a customer security questionnaire. The questionnaire asks: **“Before onboarding a vendor that handles regulated customer data, what assessments are required, who approves the exceptions, and which policies apply?”**

A plausible answer might need material from the security policy, a procurement procedure, a vendor-risk playbook, the latest exception register, and a contract. A language model on its own cannot be trusted to know which internal versions are current. Simply pasting a stack of documents into a prompt can be costly, cumbersome, and insecure. And even if the correct passage is retrieved, the model can still misinterpret it.

This is where retrieval-augmented generation (RAG) enters the story. RAG supplements a model's general capabilities with **retrieved evidence at inference time**. The architectural goal is not to make the model seem more confident. It is to deliver an answer that is *supported, authorized, timely, and useful*.

There is a broader lesson I learned while working on enterprise AI applications: retrieval quality and answer quality are different problems. Finding the right passage is only the start; verifying a claim, handling uncertainty, tracing the source, and assigning human accountability can be just as important.

This guide uses one recurring example—an enterprise **Question Assistant**—to explore six complementary architecture patterns. It also distinguishes reported implementation experience from a forward-looking design exercise. The architectures are **not** a universal maturity ladder in which each new technique replaces its predecessor.

A real-world anchor: the Question Assistant

In work on a question-answering application at Fortress, the challenge was to accelerate responses to enterprise and cybersecurity assessments without sacrificing defensibility. The work progressed from an initial RAG approach to hybrid retrieval and a separate quality-control workflow. That workflow employed **three agents backed by different LLMs**: an **Extractor**, a **Validator**, and an **Evaluator**, with human review when needed.

The Extractor was responsible for proposing an answer from the available evidence. The Validator checked whether its claims were supported. The Evaluator assessed response quality and suitability. A human reviewer remained part of the workflow for cases requiring judgment or correction.

That is a meaningful architecture distinction. **A multi-agent answer-validation system is not automatically agentic RAG**. Agentic RAG specifically gives an orchestration system the ability to decide what to retrieve next, use additional tools, and iterate until a question is adequately supported. The original validation workflow and a retrieval agent can complement each other, but they solve different problems.

Historical boundary: The initial RAG, hybrid-retrieval, three-model-agent, and human-review descriptions reflect project recollections supplied for this case study. The precise technology stack, order and dates of implementation, production metrics, and public-disclosure permissions remain to be verified. The graph, iterative agentic retrieval, Google's specific sufficiency architecture, and LLM Wiki material below are **proposed future enhancements**, not claims that Fortress implemented them.

1. Naive RAG: locate useful passages

The simplest practical architecture has two lifecycles.

Ingestion: collect documents, parse their content, separate them into useful chunks, generate an embedding for each chunk, and store those vectors with the text and relevant metadata. Useful metadata includes source ID, document version, effective date, ownership, and—critically—access restrictions.

Retrieval and generation: embed the user question with a compatible embedding model, retrieve similar passages, and provide those passages to an LLM with instructions to answer from them. The final response should point back to its source material.

What is an embedding? It is a numerical representation produced by a model so that semantically related passages can be close together in a vector space. Thus “supplier security review before approval” can be found in response to a question about “vendor cyber assessment during onboarding,” even though the words are not identical. Similarity may be computed with cosine similarity or another compatible metric. Vector databases often use approximate nearest-neighbor indexes to search large collections efficiently.

Why chunk at all? A 500-page policy manual often mixes unrelated subjects, and embedding the whole document can blur local meaning. Smaller chunks can improve relevance and reduce unnecessary prompt context; larger chunks preserve nuance and references. Chunk size is a tuning parameter, not a law. A reasonable experiment compares chunk sizes and overlap while checking whether headings, tables, and policy exceptions remain intelligible.

Failure mode: the top five closest chunks can be *plausibly related* without containing the actual answer. A citation to an irrelevant paragraph is no better than no citation. A model can also hallucinate despite being shown valid passages.

When it fits: a relatively simple document assistant with narrow domains, manageable update patterns, and evaluation data showing that first-pass similarity search is good enough.

2. Advanced RAG: combine exact terms and meaning

Enterprise language has two personalities. People ask conceptual questions—“How do we assess vendor risk?”—but also refer to exact identifiers such as TPRM-ENT-2026, contract 8472, security control codes, model names, and regulatory clauses. Pure semantic matching can miss those exact-match requirements.

BM25 is a well-established lexical ranking method. It scores how well query terms match a document or passage, considering term frequency, how uncommon a term is in the collection, and length normalization. It requires no embedding model. It is useful when exact vocabulary matters, although it does not understand paraphrases in the way semantic retrieval can.

Hybrid retrieval combines lexical and semantic results. A typical pipeline runs BM25 and vector search, merges the candidate rankings using **reciprocal rank fusion (RRF)** or calibrated score fusion, and then optionally applies a **reranker**. RRF is attractive because the two retrieval engines' raw scores often occupy incomparable scales; it combines ranked positions rather than naïvely adding scores.

A **reranker** takes a smaller candidate pool—say, 30 or 50 passages—and applies a more expensive relevance model, often a cross-encoder that examines the question and each candidate together. It then selects the strongest evidence for the LLM. Reranking can improve precision but **cannot rescue the correct policy if the first retrieval stage never found it**.

This is the ideal place to test query rewriting, policy metadata filters, language-specific analyzers, version-aware ranking, and full-document citation expansion. It is also where permissions need to be enforced **before candidate content is exposed to the model**, not after generation.

Failure mode: an apparently sophisticated retrieval chain can still blend outdated and approved policy versions, overvalue repeated terms, or retrieve a document the employee cannot access.

When it fits: a production enterprise assistant that must understand ordinary language yet reliably locate SKUs, contractual IDs, policy numbers, and technical vocabulary. For many teams, hybrid retrieval plus reranking is a better first investment than an elaborate agent architecture.

Case-study spotlight: three-model quality control

Retrieval was only one axis of the Question Assistant's design. The other was **answer assurance**.

The reported quality-control workflow placed three independently modeled functions around generated answers:

1. **Extractor agent** — proposes an answer and maps it to source passages.
2. **Validator agent** — checks individual claims against the evidence and flags gaps or contradictions.
3. **Evaluator agent** — assesses relevance, coverage, response quality, and readiness for review.

Each function used a different LLM according to the project recollection, helping avoid a single-model monoculture. This offers the possibility of independent perspectives, but independence is not guaranteed merely because model names differ. The checks must themselves be evaluated against a human-reviewed benchmark.

Human-in-the-loop review provides an escalation path for uncertain, sensitive, or disputed answers. Not every answer necessarily needs manual approval, and not every automated disagreement is a real error. A robust workflow defines escalation triggers, stores evidence and reviewer decisions, and records corrections in an auditable way.

The most useful way to tell this story is not as “three agents equals zero hallucination.” It is as a design for **defense in depth**. Retrieval increases access to relevant evidence; validation tests factual support; evaluation tests usefulness and completeness; human judgment sets the boundary for consequential decisions.

The technical animation accompanying this section should show the extractor's answer moving through validation and evaluation. When a claim lacks support, the path branches to human review rather than presenting an invented answer as fact.

3. GraphRAG: retrieve relationships, not only snippets

Suppose the employee asks: “**Which high-risk vendors supply products used in regulated workloads, and what open exceptions affect those products?**” The answer may involve a vendor directory, contracts, application inventories, assessment findings, and policy exceptions. Several snippets may each be correct but fail to reveal the relationships between them.

GraphRAG introduces structured knowledge about **entities and edges**. A vendor *supplies* a product; the product *runs in* a business application; the application *processes* regulated data; an exception *applies to* a control. The system can traverse those relationships and retrieve underlying documents for evidence.

Microsoft's GraphRAG approach goes beyond an entity graph alone: it extracts entities and relationships from text, groups them into communities, and creates community-level summaries. Different retrieval modes support local entity questions and more global questions about a whole corpus.

One caution matters: **graphs can also hallucinate** when entity extraction incorrectly links records, confuses similarly named suppliers, or treats an expired relation as current. Graph edges need provenance, update mechanisms, and security checks. It also costs time and compute to construct and refresh the graph.

When it fits: multi-document questions where the relationship between policies, assets, vendors, contracts, and risk findings is central. It is rarely the default for a straightforward FAQ.

4. Agentic RAG: retrieve iteratively

Traditional RAG frequently runs a single search and returns what it finds. An agentic retrieval workflow instead asks, “**What evidence do I still need?**”

Consider a compound question: “**For our cloud-based onboarding solution, identify the relevant security requirement, determine whether the current vendor assessment covers it, and find who approved the exception.**” One query may find the requirement but not the assessment, or the assessment but not the approver.

An agent can decompose the request, choose a source, search for the assessment, follow a referenced exception ID, query another system, and evaluate whether the evidence is sufficient. An orchestration loop may continue with a bounded number of searches before answering or abstaining.

The benefits are particularly compelling when facts are distributed among systems that index different things: CRM records, document stores, service tickets, and security systems. The risk is that a poorly constrained agent spends too long searching, follows irrelevant leads, or attempts an unauthorized access path.

A production design therefore needs clear tool permissions, per-source authorization, retrieval logs, budget and latency limits, stopping rules, and explicit abstention behavior. It should also distinguish **read-only information gathering** from agents empowered to take business actions.

When it fits: questions that require multi-step evidence gathering, source navigation, or query reformulation—and where measured improvements justify the additional operational complexity.

5. Google's sufficient-context approach: know when to keep searching

On **June 5, 2026**, Google Research described an agentic RAG framework associated with the **Gemini Enterprise Agent Platform**, emphasizing cross-corpus retrieval and an explicit **sufficient-context** assessment. Its workflow breaks complex queries into smaller information needs, searches across sources, evaluates whether the evidence is adequate, and targets missing facts with follow-up queries.

The important innovation is not that Google invented iterative retrieval. It is the deliberate inclusion of a **sufficiency check** in a system designed to avoid premature answers. The system should recognize the difference between *a relevant-looking result* and *enough evidence to complete the user's request*.

For our Question Assistant, the corresponding flow might be: locate the governing vendor policy; identify the data-sensitivity classification; find the relevant approval authority; check whether the answer has all three; then retrieve again if the approver is missing.

Google reported **up to 34% higher accuracy compared with standard RAG on its evaluated factuality datasets**. This is a research result in a particular evaluation setup—not a universal enterprise performance claim, a guarantee, or a result from the Fortress application.

In implementation, a sufficiency score is not a substitute for truth. The assistant can be confidently wrong about the adequacy of the retrieved evidence. Human-audited examples and tests for both false confidence and unnecessary extra searching remain essential.

When it fits: complex multi-source questions where incomplete answers are costly and targeted follow-up retrieval can be measured to improve answer completeness.

6. Karpathy's LLM Wiki: make knowledge accumulate

In an **April 4, 2026** idea file, **Andrej Karpathy** proposed a pattern he calls **LLM Wiki**. Instead of having an LLM rebuild knowledge from raw documents for each query, an LLM agent incrementally maintains an interconnected, human-readable set of knowledge pages.

The original proposal has three layers: **raw, authoritative sources; a maintained wiki; and a schema or instruction file** that governs ingestion, querying, and maintenance. New sources update topic pages, entity profiles, summaries, backlinks, and contradiction notes. Periodic quality checks—Karpathy calls them *linting*—look for stale information, broken relationships, and missing coverage.

In an enterprise Question Assistant, one could imagine vendor pages, policy pages, control explanations, and lists of open exceptions being continually refreshed from approved sources. A query reads the accumulated knowledge first, then follows provenance links to verify details as needed.

This is an **idea pattern, not proof that a wiki universally outperforms RAG**. An LLM-maintained summary can drift, accidentally merge contradictory facts, or become stale. For enterprise use it would need lifecycle governance, approved-source versioning, review workflows, access-aware indexing, and clear separation of AI-created summaries from original evidence. It can **complement hybrid RAG** rather than replace it.

When it fits: recurring research questions, rapidly evolving internal knowledge, and information domains where curating relationships across time is valuable enough to support a maintained knowledge layer.

How I would choose an architecture in 2026

My first design question would not be “Which new RAG pattern should we deploy?” It would be “What types of questions must this system answer, with what reliability, response time, and authorization constraints?”

Requirement	Starting approach	Why
Simple, well-curated documentation Q&A	Baseline RAG	Low operational complexity if measured quality is adequate
Exact IDs plus natural-language intent	Hybrid BM25 + vectors	Balances literal precision and semantic recall
Stronger evidence ranking	Hybrid + reranking	Refines an already good candidate set
Connected asset/vendor/contract reasoning	GraphRAG pilot	Traverses explicit relationships
Multi-step, multi-system investigations	Bounded agentic RAG	Iteratively seeks missing evidence
Incomplete evidence is a frequent failure	Sufficiency-aware retrieval	Checks coverage before generating
Recurring questions over evolving organizational knowledge	LLM Wiki experiment	Maintains a reusable, linked knowledge layer

Requirement	Starting approach	Why
Consequential or disputed generated claims	Validator/evaluator + human escalation	Adds quality governance across any retrieval pattern

This is not a set of mutually exclusive choices. A well-designed assistant might combine hybrid retrieval, graph traversal for selected questions, an agentic planner for difficult queries, and independent answer validation. Complexity should be added only when evaluation shows it addresses a real failure mode.

What must be measured, not merely demonstrated

An architecture diagram does not prove enterprise readiness. A credible evaluation program needs a **versioned, held-out question set** with approved answers and source evidence, including adversarial cases and documents with deliberate conflicts. At a minimum I would track:

- **Retrieval recall@k**: did the retrieval candidates contain the evidence needed to answer?
- **Ranking quality**: how often did the most useful passage reach the generation context?
- **Answer correctness and faithfulness**: was the answer accurate and fully supported by the cited sources?
- **Abstention quality**: did the system appropriately decline to answer when evidence was absent or contradictory?
- **Human escalation rates and reviewer corrections**: how often did the system require intervention, and why?
- **Latency, token and retrieval cost, and operational reliability**: can it work economically at scale?
- **Authorization leakage tests**: can one employee retrieve or infer material restricted to another role?

For graph and agentic systems, I would additionally test relationship correctness, unnecessary tool calls, stopping behavior, and whether authorization is enforced **at each retrieval step**. For a maintained wiki, I would track update lag, provenance coverage, edit review, staleness, and error propagation.

If I were redesigning the Question Assistant today

I would start by establishing a strong, auditable retrieval baseline: approved-document ingestion, document and chunk provenance, permissions at retrieval time, hybrid search, and measured reranking. I would retain the extractor–validator–evaluator design as a separate quality-assurance layer, with independent human calibration and escalation policies.

Next, I would run limited experiments. I would use a graph only for questions that genuinely require relationship traversal, and give a retrieval agent a narrow set of read-only tools for multi-step questions. I would test a sufficiency check on cases that currently fail because the assistant stops with partial evidence. A living wiki could be piloted for repeated, high-value topics, with human-approved updates and original sources retained as the authority.

All of this would be governed by one principle: **business value comes from better decisions and dependable workflows—not from the number of agents, indexes, or architecture labels in the system.**

Our Question Assistant story began with the practical problem of answering assessment questions accurately. It leads to a larger enterprise AI principle: retrieval provides evidence; orchestration gathers what is missing; validation challenges unsupported claims; and human governance decides what can be trusted. The future of RAG is not a single database or framework. It is a disciplined architecture for turning organizational information into trustworthy action.

Sources and further reading

1. Google Research, “Unlocking dependable responses with Gemini Enterprise Agent Platform's Agentic RAG,” June 5, 2026. <https://research.google/blog/unlocking-dependable-responses-with-gemini-enterprise-agent-platforms-agentic-rag/>
2. Andrej Karpathy, “LLM Wiki,” original idea file, April 4, 2026. <https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f>
3. Microsoft GraphRAG documentation. <https://microsoft.github.io/graphrag/>
4. Elastic, “Hybrid search” and ranking documentation. <https://www.elastic.co/docs/solutions/search/hybrid-search> and <https://www.elastic.co/docs/solutions/search/ranking>

Editorial note: This case study is based on project recollections and public technical sources. Validate project chronology, implementation details, metrics, company naming, and disclosure permissions before publishing. Illustrative examples are not customer records.